TIBCO eFTL™ Development

Software Release 3.3
October 2017

Two-Second Advantage®

Important Information

Contents

About this Product 5
TIBCO Documentation and Support Services 6
API Overview 7
API Reference Documentation 8
Sample Programs 9
Client Platform Support and Requirements Reference 10
Java Programmer's Checklist 11
Android Programmer’s Checklist 12
Windows .NET Programmer’s Checklist 13
iOS and Objective C Programmer’s Checklist 14
JavaScript for Browsers Programmer’s Checklist 15
JavaScript for Node.js Programmer’s Checklist 16
C Programmer’s Checklist 17
Restrictions on Names 18
Reserved Names 18
Length Limit 18
Programming an eFTL Application 19
Listeners and Server Callbacks 19
Persistence 21
FTL Last-Value Durables 22
Interaction with EMS 23
Push Notifications 24
Programming Push Notifications 24
Coordination Forms 25

About this Product

TIBCO® is proud to announce the latest release of TIBCO eFTL ™ software.

This release is the latest in a long history of TIBCO products that leverage the power of Information Bus® technology to enable truly event-driven IT environments. To find out more about how TIBCO eFTL software and other TIBCO products are powered by TIB® technology, please visit us at www.tibco.com.

Product Editions

TIBCO eFTL and related products are now available in a community edition and an enterprise edition.

TIBCO eFTL - Community Edition is ideal for getting started with eFTL, for implementing application projects, including proof of concept projects, for testing, and for deploying applications in a production environment. Although the community license limits the number of production processes, you can easily upgrade to the enterprise edition as your use of TIBCO eFTL expands.

The community edition is available free of charge. It is a full installation of the TIBCO eFTL product, with the following limitations and exclusions:

- Users may run up to 1000 mobile clients in a production environment, per corporate entity.
- Users do not have access to TIBCO Support. However, you can use TIBCO Community as a resource (https://community.tibco.com).

TIBCO eFTL - Community Edition is compatible with both the enterprise and community editions of TIBCO FTL®.

TIBCO eFTL - Enterprise Edition is ideal for all application development projects, and for deploying and managing applications in an enterprise production environment. It includes all features presented in this documentation set, and access to TIBCO Support. Choose the enterprise edition for production deployments with more than 1000 mobile clients, and for enterprise monitoring using customizable status dashboards.

TIBCO eFTL - Enterprise Edition uses the enterprise edition of TIBCO FTL and includes a license for it.

TIBCO Documentation and Support Services

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit https://docs.tibco.com.

Product-Specific Documentation

The following documents for this product can be found on the TIBCO Documentation site:

- *TIBCO eFTL Concepts*
- *TIBCO eFTL Administration*
- *TIBCO eFTL Development*
- *TIBCO eFTL Installation*
- *TIBCO eFTL API Reference* (HTML only)
- *TIBCO eFTL Release Notes*

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit http://www.tibco.com/services/support.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at https://support.tibco.com.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to https://support.tibco.com. If you do not have a user name, you can request one by clicking Register on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to https://community.tibco.com.

API Overview

The TIBCO eFTL API is a compact version of the TIBCO FTL API, small enough for mobile applications and browser applications. It includes everything you need to exchange messages, and it omits everything else.

Functionality

Programs use the API for these operations:

- Connecting to an eFTL server
- Publishing messages (one-to-many)
- Subscribing to messages (one-to-many) and filtering with content matchers
- Constructing and unpacking messages

Asynchronous Callback Architecture

API calls return quickly, which reflects a requirement of mobile device applications. All API calls that depend on a TIBCO eFTL server are asynchronous. When the server responds, the TIBCO eFTL library invokes a user-defined callback method.

Message Data Types

The field data types within eFTL messages are similar to those in FTL messages, except that eFTL messages do not support the opaque and inbox types.

However, the field data types within eFTL messages are *not* similar to those in EMS messages. For details of message translation, see “Message Translation: TIBCO eFTL and TIBCO EMS” in *TIBCO eFTL Concepts*.

API Reference Documentation

TIBCO eFTL API reference documentation is available only in HTML format.

The documentation set includes API reference resources for each supported programming language or framework.

Sample Programs

Each software development kit (SDK) includes sample programs for the corresponding development platform. After extracting the SDK from its archive file, you can locate the samples directory at `eftl/`*`version_number`*`/samples/`*`platform`*.

Client Platform Support and Requirements Reference

TIBCO eFTL software supports these platforms for application development and deployment.

For platform version requirements, see the file `readme.txt`.

Client Platform	API Support
Android	Java API
iOS (Apple)	Objective C API
Web Browsers	JavaScript API
Node.js	JavaScript API
Windows (Microsoft)	.NET API C Golang
Linux macOS	C Golang

Java Programmer's Checklist

When developing eFTL applications in Java for *platforms other than Android*, remember these steps.

Environment

- JDK 1.8

Code

Ensure that the following library is in the `CLASSPATH`:

- `*TIBCO_HOME*/eftl/*version*/sdks/java/lib/tibeftl.jar`

Compile

None.

Run

- Ensure that the TIBCO FTL realm server is running, and that each TIBCO eFTL server process can connect to it.
- If your application communicates over EMS channels, ensure that the TIBCO EMS server is running, and that each TIBCO eFTL server process can connect to it.
- If your application uses FTL durables, ensure that persistence stores are configured, running, and reachable.
- Ensure that the TIBCO eFTL server is running, and that each client process can connect to it.

Android Programmer’s Checklist

When developing eFTL applications for Android platforms, remember these steps.

Environment

- Use the most recent release of Android Studio.

Code

Add the following library to your project:

- `TIBCO_HOME/eftl/version/sdks/java/lib/tibeftl.jar`

Compile

None.

Run

- Ensure that the TIBCO FTL realm server is running, and that each TIBCO eFTL server process can connect to it.
- If your application communicates over EMS channels, ensure that the TIBCO EMS server is running, and that each TIBCO eFTL server process can connect to it.
- If your application uses FTL durables, ensure that persistence stores are configured, running, and reachable.
- Ensure that the TIBCO eFTL server is running, and that each client process can connect to it.

Windows .NET Programmer’s Checklist

When developing eFTL applications for Windows or for Windows tablet devices, remember these steps.

Environment

- Set up the environment for .NET Framework 4.6.1.

  Install the .NET framework *first*, before installing TIBCO eFTL software. If the .NET framework is present, then installing TIBCO eFTL automatically registers the appropriate assemblies in Microsoft’s general assembly cache (GAC).
- Use the `setup` script in the `samples` directory to set environment variables.
- In addition, ensure that the `PATH` environment variable includes the .NET Framework SDK (`%windir%\Microsoft.NET\Framework64\`*`dotnet_version`*).

Code

- Use the Visual Studio 2015 programming environment.
- Import this assembly:
  - *`TIBCO_HOME`*`/eftl/`*`version`*`/sdk/dotnet/bin/TIBCO.EFTL.dll`

Compile

- Compile with any .NET compiler.

Run

- Ensure that the TIBCO FTL realm server is running, and that each TIBCO eFTL server process can connect to it.
- If your application communicates over EMS channels, ensure that the TIBCO EMS server is running, and that each TIBCO eFTL server process can connect to it.
- If your application uses FTL durables, ensure that persistence stores are configured, running, and reachable.
- Ensure that the TIBCO eFTL server is running, and that each client process can connect to it.

iOS and Objective C Programmer’s Checklist

When developing eFTL applications for Apple iOS platforms, remember these steps.

Environment

- Add `TIBCO_HOME/eftl/version/sdk/ios/eFTL.framework` to your project.
- Add these Xcode libraries and frameworks to the build phases of your project:
  - `libicucore.tbd`
  - `CFNetwork.framework`
  - `Security.framework`
- Set the **Other Linker Flags** build setting to include `-ObjC`.

Code

- Import the TIBCO eFTL header file, `eftl/eftl.h`, which in turn includes other header files.

Compile

None.

Run

- Ensure that the TIBCO FTL realm server is running, and that each TIBCO eFTL server process can connect to it.
- If your application communicates over EMS channels, ensure that the TIBCO EMS server is running, and that each TIBCO eFTL server process can connect to it.
- If your application uses FTL durables, ensure that persistence stores are configured, running, and reachable.
- Ensure that the TIBCO eFTL server is running, and that each client process can connect to it.

JavaScript for Browsers Programmer’s Checklist

When developing eFTL JavaScript programs for browser platforms, remember these steps.

Code

JavaScript programs must reference the TIBCO eFTL library package, `*TIBCO_HOME*/eftl/*version*/sdks/javascript/lib/eftl.js`.

Run

- Ensure that the TIBCO FTL realm server is running, and that each TIBCO eFTL server process can connect to it.
- If your application communicates over EMS channels, ensure that the TIBCO EMS server is running, and that each TIBCO eFTL server process can connect to it.
- If your application uses FTL durables, ensure that persistence stores are configured, running, and reachable.
- Ensure that the TIBCO eFTL server is running, and that each client process can connect to it.
- If you run a secure eFTL server, which listens using `wss` protocol, ensure that the client browser trusts the server certificate of the TIBCO eFTL server; see “Clients Trust the Server” in *TIBCO eFTL Administration*.
- The Compatibility View feature of Internet Explorer interferes with correct operation of JavaScript programs. Instruct application users to disable this feature.

  To disable the Compatibility View feature, click **Tools > Compatibility View Settings** to open the settings dialog. Clear all the check boxes.

JavaScript for Node.js Programmer’s Checklist

When developing eFTL JavaScript programs for Node.js environments, remember these steps.

Code

- JavaScript programs must require the TIBCO eFTL library package, `*TIBCO_HOME*/eftl/*version*/sdks/javascript/lib/eftl.js`.

  ```
  var eftl = require('eftl')
  ```

- If you run a secure eFTL server, which listens using `wss` protocol, the program must register the server certificates that the application trusts. If the program does not register server certificates, then it trusts any certificate that the server present. (The JavaScript method that registers certificates applies *only* when running in a Node.js environment. It is not effective when running in a browser.)

Run

- eFTL programs require that the WebSocket module is installed in the Node.js environment.

  Install it using this command line:

  ```
  npm install ws
  ```

- Ensure that the TIBCO FTL realm server is running, and that each TIBCO eFTL server process can connect to it.
- If your application communicates over EMS channels, ensure that the TIBCO EMS server is running, and that each TIBCO eFTL server process can connect to it.
- If your application uses FTL durables, ensure that persistence stores are configured, running, and reachable.
- Ensure that the TIBCO eFTL server is running, and that each client process can connect to it.
- Ensure that the file `eftl.js` is available in the environment.

C Programmer’s Checklist

When developing eFTL applications in C, remember these steps.

The eFTL C API is available as platform-specific binary libraries. It is also available in source form, so you can compile it for other platforms.

Environment

Use the `setup` script in the `samples` directory to set environment variables.

Code

- Include the TIBCO eFTL header file, `tib/eftl.h`.

Compile

Compile programs with an ANSI-compliant C compiler.

Use the compiler flags in the `Makefile` in the `samples/c/examples` directory.

Link

On UNIX platforms, use the linker flags `-ltibeftl`.

On Windows platforms, link the corresponding C library file, `tibeftl.lib`.

Run

- On Windows platforms, the `PATH` variable must include the `bin` directories to use the DLL files for TIBCO eFTL and TIBCO FTL.
- On UNIX platforms, the `LD_LIBRARY_PATH` variable must include the `lib` directories to use the library files for TIBCO eFTL and TIBCO FTL.
- Ensure that the TIBCO FTL realm server is running, and that each TIBCO eFTL server process can connect to it.
- If your application communicates over EMS channels, ensure that the TIBCO EMS server is running, and that each TIBCO eFTL server process can connect to it.
- If your application uses FTL durables, ensure that persistence stores are configured, running, and reachable.
- Ensure that the TIBCO eFTL server is running, and that each client process can connect to it.

Restrictions on Names

Names and other literal strings must conform to the restrictions in the following topics.

Reserved Names

All names that begin with an underscore character (_) are reserved. It is illegal for programs to define formats, fields, endpoints, and applications with these reserved names.

You may use the underscore character elsewhere, that is, where it is *not* the *first* character of a name.

Length Limit

All names are limited to a maximum length of 256 bytes.

This limit includes names of formats, fields, endpoints, and applications. It also extends to string values in content matchers.

If you use UTF-8 characters that are larger than one byte, remember that the limit restricts the number of bytes in the name, rather than the number of characters.

Programming an eFTL Application

When programming an eFTL application, use the steps of this task to guide the program structure.

Some programs require only a subset of these structural steps.

Procedure

1. Define Connection Callbacks
   Every program must define the four ConnectionListener event methods:
   - Connect
   - Disconnect
   - Reconnect
   - Error
2. Define Subscription Callbacks
   Programs that subscribe must define the three SubscriptionListener event methods:
   - Messages
   - Subscribe
   - Error
3. Define completion callbacks.
   Programs that publish define the two CompletionListener event methods:
   - Completion
   - Error
4. Connect to an TIBCO eFTL Server
   a) If the server uses TLS security, the application must register trust in the server's certificate.
   b) Every program must call the connect method to establish a connection.
      You may include user name credentials in the **`url`** argument. For syntax, see the API documentation. When using the WSS protocol, this information travels encrypted.
5. Sending Messages

   Programs that publish must complete both of these substeps.
   a) Construct outbound messages.
   b) Call publish the method.
6. Receiving Messages

   Programs that subscribe must call the subscribe method.
7. Clean-Up

   Consider whether it is appropriate for the program to close subscriptions or durable subscriptions.

   Every program must call the disconnect method to disconnect from the server.

Listeners and Server Callbacks

Client requests to the eFTL server are asynchronous. To ascertain the outcome of a request, clients must supply a callback listener.

A *send* call is a request that the eFTL server send a message for the client. A send call returns before the server fulfills the request. To receive information about the outcome of the send request, the application must supply a *completion listener* as an argument to the send call.

It is good practice to supply a completion listener when using persistence features or when interacting with an EMS server.

In earlier versions of the TIBCO eFTL API, send calls did not require a completion listener. It is good practice to update application programs to supply completion listeners whenever the outcome a send call is important to the application.

A *subscribe* call is a request that the eFTL server subscribe to messages for the client. A subscribe call returns before the server fulfills the request. To receive messages and information about the status of the subscription, the application must supply a *subscription listener* as an argument to the subscribe call.

A *connect* call is a request that the eFTL server establish a connection for the client. A connect call returns before the server fulfills the request. To receive information about the status of the connection, the application must supply a *connection listener* as an argument to the connect call.

Persistence

Your eFTL applications can use the persistence features of either the FTL persistence server or the EMS server.

Coordination

Persistence depends on proper configuration of the eFTL server channel, and of the FTL persistence server or EMS server, corresponding to the channel's message infrastructure. Developers must coordinate with administrators to agree on the requirements for this configuration. Complete the appropriate coordination forms together.

Durable Subscriptions

A durable subscription is a subscription that persists even when the application is not running. The FTL persistence server or the EMS server maintains the durable subscription on behalf of the application.

Unsubscribe

The unsubscribe call deletes the durable subscription. The server discards all messages that it holds for the subscription.

FTL Persistence

The FTL channel must be configured for persistence. (See "Configuring Persistence for an FTL Channel" in *TIBCO eFTL Administration*.)

eFTL programs can subscribe to *static* durables of any type: standard, shared, or last-value durables.

However, eFTL programs can create and subscribe to *dynamic* durables only if the template specifies the type as a *standard* durable. For shared and last-value semantics, use static durables.

When you supply a durable name in the subscribe call, the call resolves it to a unique durable name in two phases:

1. **Static** If the channel definition includes a static durable mapping, and it maps the durable name to a static durable in the channel's persistence store, then the result is a subscription to that static durable.

   Subscriber name mapping is an advanced topic.

2. **Dynamic** If the first phase did not result in a valid static durable, and the channel definition includes a dynamic durable template, then the result is a subscription to a dynamic durable with a unique concatenated durable name (see the section "Concatenated Durable Names," which follows).

These anomalous situations trigger the subscription listener's error callback:

- You supply a durable name but the FTL channel is *not* configured for persistence.
- The FTL channel *is* configured for persistence but you do *not* supply a durable name.
- The subscribe call cannot resolve the durable name you supply to either a static or a dynamic durable.

See also Last-Value Durables.

EMS Persistence

The EMS server must be configured to support concatenated durable names.

Every subscribe call results in a durable subscription to a concatenated durable name (see "Concatenated Durable Names").

Concatenated Durable Names

The client library forms a unique concatenated durable name by prefixing the client ID string and a period (`.`) character to the program's durable name (as supplied in the subscribe call).

The concatenated name must not exceed the maximum name length. See Length Limit.

Client ID

The client ID indicates the particular process instance of the application program that created the durable subscription. Usually this ID represents either the user of the application or a specific mobile device. Two client processes cannot use the same client ID at the same time.

You may supply the client ID in the connect call. For long-term persistence, it is good practice to explicitly supply a client ID.

If you do not supply a client ID, the connect call assigns a unique client ID. However, such assigned ID strings can provide only short-term persistence. If the client process disconnects from the eFTL server and then reconnects, an assigned ID preserves the durable subscription through that time gap. However, if the application restarts, then the durable subscription is not available.

FTL Last-Value Durables

Subscribing to an FTL last-value durable requires a content matcher.

TIBCO eFTL supports only static last-value durables.

The FTL admininstrator configures the key field name.

In the client program the content matcher in the subscribe call must match a specific string value of that key field. For example, if the key field is `"flight_num"`, then the content matcher `{"flight_num":"954"}` subscribes to updates for a specific flight.

Interaction with EMS

When developing eFTL applications that could communicate over EMS channels, use the `_dest` field for addressing.

The `_dest` field contains a string that corresponds to a JMS topic name; for example `NYSE.IBM`. EMS channels of the eFTL server translate between these two representations.

- Code eFTL publishers to set the `_dest` field when constructing messages.
- Code eFTL subscriber content matchers to match against the `_dest` field, and not against any other fields; for example, `{"_dest":"NYSE.IBM"}`.

For background information, see “Message Translation: TIBCO eFTL and TIBCO EMS” in *TIBCO eFTL Concepts*.

Push Notifications

Mobile devices can receive push notifications that inbound messages are waiting at the server.

If push notifications are enabled, the server retains the client's subscriptions, even when the client application is disconnected from the server. When the application reconnects, the server delivers the backlogged messages to the application.

If a message arrives while the client is disconnected, the server requests that the platform vendor's push notification service (PNS) send a silent notification to the device. Upon receiving a notification, the device runs the application's notification callback. (*Silent notifications* do *not* implicitly display badges or text to the device user. However, your notification callback can explicitly implement these or other appropriate behaviors.)

Notifications are available through the following push notification services:

- Apple Push Notification Service (APNS) for iOS platforms
- Google Cloud Messaging (GCM) for Android platforms

Push notifications are an optional feature of TIBCO eFTL software.

To use push notifications, the developer must code the application appropriately, and the administrator must configure the server appropriately. (See "Configuring Push Notifications" in *TIBCO eFTL Administration*.)

Programming Push Notifications

To implement push notifications for an eFTL application, complete these steps in your mobile application code.

Prerequisites

- Your application runs on a mobile platform that supports push notification.
- Your application already has an ID and authentication with the platform vendor's push notification service (PNS).

Procedure

1. Code a notification callback.

   Conform to the requirements of the vendor's push notification service.
2. Request a notification token from the PNS.
   The token uniquely represents your application running on a specific mobile device. The token also represents your application within the device's operating system.
3. Supply the notification token as a property in the eFTL client connect call.
4. Create eFTL sbuscriptions as usual.
5. Disconnect from the server when the application exits or transitions to a background state.
   While the client is disconnected, the server maintains the client's eFTL subscriptions, and sends a push notification whenever an eFTL message arrives.
6. Reconnect to the server when the application transitions to a foreground state.
   The server automatically delivers any eFTL messages that are waiting for the client.

Coordination Forms

Coordination forms help developers and administrators to agree upon application details and to document those details.

You can duplicate these PDF forms, renaming the copies, and complete them online. Alternatively, you can print them and complete them on paper.

- TIBCO eFTL Application Coordination Form
- TIBCO eFTL EMS Channel Coordination Form
- TIBCO eFTL FTL Channel Coordination Form
- TIBCO eFTL Server Coordination Form